# Allocation Manager Requirements Document

Scott Jackson
Pacific Northwest National Laboratory
Scott.Jackson@pnl.gov

# 1 Introduction

## 1.1 Purpose

This document details the functionality requirements for the Allocation Manager component to be produced by the Scalable Systems Software (SSS) Center.

## 1.2 Intended Audience

This document is primarily intended for SSS developers (particularly those responsible for aspects of the resource management system) as well as managers and system administrators of terascale computer centers around the nation (particularly those at DOE sites).

## 1.3 Scope

The creation of this requirements document has several objectives. It is useful for project management, both in schedule planning and to provide a progress metric. It is useful for other component developers to understand what the key features are and how to interact with the Allocation Manager. Furthermore, it can be presented to the customers and reviewed to ensure core requirements are satisfied and to elicit feedback with respect to desired capabilities and interface specifications.

## 1.4 Overview

We propose to develop a dynamic allocation manager which will interoperate within a resource management system (composed of a resource manager, a scheduler, and optionally a meta-scheduler) to manage the allocation of CPU and other resources to projects and users. The target operating environment is that of UNIX-based high-performance computing systems.

## 1.5 Organizational Context

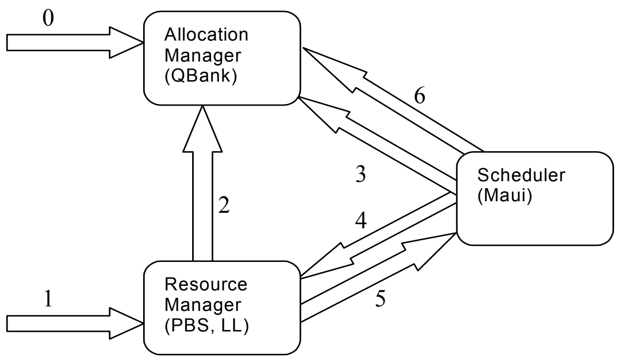
This effort is funded by the U.S. Department of Energy (DOE) as part of the Scientific Discovery through Advanced Computing (SciDAC) Initiative. The software infrastructure vision of SciDAC is for a comprehensive, scalable, robust, portable, and fully integrated suite of systems software and tools for the effective management and utilization of terascale computational resources by SciDAC applications. The Scalable Systems Software Center for its part is responsible to provide an integrated suite of components including resource scheduling, usage accounting and user interfaces.

## 2 General Description

### 2.1 System Functionality

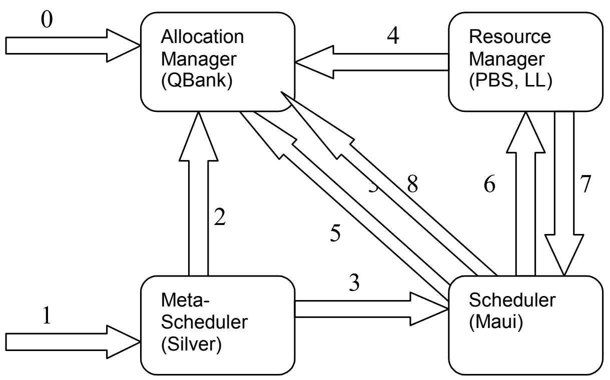
An allocation management system provides a means to fairly distribute computing resources (processors, memory, disk) to the various users or projects that need them. Much like a bank an allocation manager (or allocation bank) associates a cost to computing resources and allows resource tokens to be allocated to users and projects. Debits are made (in the form of withdrawals) against these allocations when compute resources are used. An allocation manager provides an administrative interface supporting familiar operations such as deposits, withdrawals, transfers and refunds. Full accounting is made of resource utilization. It must provide balance and usage feedback to users, managers, and administrators.

A dynamic allocation manager interfaces with other allocation managers, schedulers, resource managers, meta-schedulers, information services and other external services. Figures 1a and 1b show examples of two typical interaction sequences.



0. Create accounts, make deposits, etc.
1. User submits a job to the resource manage
2. A balance check is made at queue time
3. A hold or reservation is placed
4. The job is started
5. The job completes
6. The reservation is removed and a final withdrawal is made against the user's allocation.

**Figure 1a. Local site allocations and job scheduling flow.**



0. Create accounts, make deposits, etc.
1. User submits a job to the meta scheduler
2. A quote could guarantee a given rate
3. The job is staged to the local scheduler
4. A balance check is made at queue time
5. A hold or reservation is placed
6. The job is started
7. The job completes
8. The reservation is removed and a final withdrawal is made against the user's allocation.

**Figure 1b. Remote site allocations and job scheduling flow.**

The allocation management system should be secure, scalable, portable, fault tolerant, reliable, and easy to use. It must maintain a persistent and queriable record of resource consumption and bank transactions. It should support flexible charging algorithms, reservations, expiring allocations, machine access controls, and multi-level authorization. It should also facilitate and enable meta-computing.

## 2.2 Similar Systems

### 2.2.1 QBank

QBank is a dynamic project accounting and allocation management system developed at PNNL. It provides a versatile means of fairly distributing an organization's computational resources (processors, memory, disk) to the various users or projects that have access to them. It supports a reservation mechanism allowing a hold to be placed on a user's resource tokens at the beginning of a job, thereby preventing overdrafts on the account. Resource allocations can have activation and expiration dates, and may be valid toward arbitrary users, and machines. It is written in Perl and uses Perl DBI to interface with a RDBMS backend.

### 2.2.2 SNUPI

SNUPI, the System, Network, Usage and Performance Interface, was created by NPACI/SDSC to provide an interface for resource utilization reporting for heterogeneous computer systems, including Linux clusters. SNUPI provides data collection tools, recommended RDBMS schema design, and Perl-DBI scripts suitable for portal services to deliver reports at the system, user, and job level for heterogeneous systems across the enterprise. Data collection can be enabled for process accounting (pacct), system activity reporting (sar), batch system accounting, and project accounting in a periodic post-processing fashion.

### 2.2.3 NIM

NIM, the NERSC Information Management System, is a resource allocation and tracking system developed by NERSC to manage account creation and usage tracking. It is a distributed, fault-tolerant web-based system implemented via PHP as a front end to a RDBMS. It is notable for it's nice web-based query interface, its role-based authorization, its implementation of a formal electronic process to request allocations, and the ability to delegate management responsibility to the organization owning the resource. Like SNUPI, there is a lag between resource usage and account updates.

## 2.3 User Characteristics

The primary intended users of the Allocation Manager will be the managers, system administrators and users of terascale computing facilities. It is expected that the system administrators will have a good understanding of resource management system concepts and require sophisticated functionality while the users will be more focused on utilizing the computational systems to achieve scientific results and will be more interested in an easy to use interface.

## 2.4 User Objectives

This section will describe the set of objectives and requirements for the system from the user's perspective. It will include feedback from surveys and a "wish list" of desirable characteristics, along with more feasible solutions that are in line with the business objectives.

- Provide a simple, flexible interface for users to track their allocation and administrators to adjust allocations on both a per user and arbitrarily defined per group basis – Brett Bode (Ames)
- Must allow tracking and accounting on a per job basis and on a per process basis – Brett Bode (Ames)
- Must allow for site dependent charging schemes for the full spectrum of resources – Brett Bode (Ames)
- Accounting system: logs job info – start, stop, resources requested vs used, exit status, project info – ANL
- User DB: Account information, allocations for users, project allocation – ANL
- Usage Reports: Summary of usage by various parameters – ANL
- Hierarchical accounts (nestable projects) – CHPC, ORNL
- Integrate with DCE – ORNL
- For auditability must be able to disable any ability to modify or remove transaction records (bank.accounting = strict) – ORNL
- It might be that an account can have an "income" associated with it and it is automatically credited with this amount at the beginning of each funding cycle – though this would not be necessary with validity periods – ORNL
- Perhaps there should be a separate job table, updateable from sources such as llsummary and rmsquery – ORNL
- The ability to request allocations through a formal process and provision of an electronic interface so that funding managers can access and participate in the allocation process online – ORNL/NERSC
- The ability to delegate management responsibility to the organization which owns the resource – NERSC
- The ability to delegate management responsibility is very important. This would best be hierarchical in nature. For example, a division manager would appear in the data structures once and have the ability to manage everything under that point in the hierarchy. This just makes the management of the system easier. – LLNL

# 3 Functional Requirements

## 3.1 Usage Accounting / Resource Utilization Tracking

The allocation manager must be able to track the resources used by each job and store this information in a persistent data store for later retrieval. By utilizing the query interface, managers should be able to produce reports detailing the system resources used by users and projects on their systems over arbitrary time intervals.

The accounting system should be able to accurately track the full spectrum of consumable resources such as node or I/O wall hours and also the use of specialized resources such as graphics or software licenses

### 3.1.1 Reports

Periodic reports should be able to be generated showing project allocations and resource utililization for that period.

### 3.1.2 Audit log (transaction log)

An audit record should be logged every time a class(table) is modified. There should also be a capability for a user to add an explicit comment to the audit table.

It may also be desirable to be able to preserve the historical state of the bank. This could be used to generate bank statements indicating past balances. Previous state can be compared with the transaction history to identify and correct database corruption. It could even be used to revert to a previous state of the bank (unless installed with accounting=strict). This could be enabled or disabled (for performance and storage space reasons).

## 3.2 Support familiar bank operations

It should support familiar operations such as deposits, withdrawals, transfers, and refunds. It should provide balance and usage feedback to users, managers and administrators.

### 3.2.1 Manage accounts (projects)

Add, query, modify or remove accounts. Accounts may be deactivated.

### 3.2.2 Manage users

Add, query, modify or remove users.

#### 3.2.3 Manage account members (subaccounts).

Add, query, modify or remove account members. Subaccounts may be individually disabled such that no scheduler transactions (reservations, withdrawals) can be made to that account on behalf of that user.

#### 3.2.4 Manage allocations

Make deposits, withdrawals, refunds and transfers against the resource allocations. Users, managers and administrators will need to obtain the account balance, machine access lists, expiration times and other information about the allocations.

#### 3.2.5 Manage groups

Add, query, modify or remove groups and group members. User and machine groups will almost certainly be required. Accounts might also be grouped.

#### 3.2.6 Manage machines (systems)

Add, query, modify or remove machines. Allocations can be valid toward machines or groups of machines.

### 3.3 Multiple users per account and multiple accounts per user

Users may given separate allocations under different projects with different resource access privileges.

### 3.4 Default Accounts

Each user may be assigned a default account to which withdrawals and reservations will be made when no account is specified.

### 3.5 Machine access lists

Users within an account may be given different allocations valid toward arbitrary groups of machines.

### 3.6 Allocation validity periods (Expiring allocations)

All allocations may be given an activation date and expiration date to define the period in which they may be consumed (implementing a use it or lose it policy). Multiple allocations with different validity periods may exist simultaneously. This feature can be used to assist users to meet a target usage distribution and prevent year-end resource exhaustion.

## 3.7 Reservations

The allocation manager should support the concept of reservations to prevent overdrafts on the account. Before a job runs, the bank will attempt to place a reservation or hold (make a pending withdrawal) on the account in behalf of the requesting user. Subsequent jobs will also place reservations while the available balance (balance-reservations) allows. When a job completes, the reservation is removed and the actual withdrawal is made to the account. This procedure ensures that jobs will only run as long as they have sufficient reserves. If the user does not have sufficient funds, the job may be deferred until additional funds are deposited into the user's account. This capability requires that when a job is submitted, the user must specify how much of the resources he expects the job to use and an upper limit for the duration of the job as well as the account which should be debited. Using reservations is optional.

## 3.8 Quotations

The ability to obtain a quote is important to support a meta-scheduling environment where it may be useful to determine how much it will "cost" to run a particular job with the specified requirements. A quote request can guarantee a charge rate based on the projected wallclock time and other parameters. Support will be build into the allocation manager to support a guaranteed quote mechanism.

## 3.9 Earliest credit expenditure

An account containing resource tokens with varying expiration dates will automatically satisfy all debit requests using the tokens with the earliest expiration date.

## 3.10Shared allocations

The allocation bank should support the capability to have all users within an account jointly share a common resource pool. This may be implemented as a pool common to all members within an account (KITTY) or as arbitrarily defined groups of users.

## 3.11Multi-level administration

It will be necessary to give different functional access privileges to different users. Bank administrators, account administrators, and regular users will require different authorization levels to perform various tasks. There should probably be view only permissions grantable to managers or user services personnel.

## 3.12 Flexible charging

The allocation manager should support a flexible and customizable mechanism for charging for resource utilization. By default, the withdrawal amount will be calculated by multiplying the number of processors used by the number of wallclock seconds taken by the job. Besides CPU, a resource supplier may charge based on the amount of memory, disk, network bandwidth used, or virtually any other consumable resource. When resources are shared, such as multiple jobs sharing CPUs on an SMP system, consumption rate charging can be used to prorate the charges according to the percentage of actual shared resources consumed. A job can be charged different static multipliers depending on quality of service requested, class, node type, or which machine it ran on. The charge algorithm should be externalized, permitting dynamic charging to be applied such as charging different rates according to time of day or week, dynamic price adjustment according to load or queue backlog, a query to an external information service, or a cached second-price auction result.

## 3.13 Peer to peer communications

A peer-to-peer communication mechanism should be implemented to carry out the exchange interactions necessary when dealing with external sites and systems.

## 3.14 Hierarchical accounts

It should support nested account hierarchies where accounts may have parent-child relationships with other accounts. In such a configuration, a deposit mask may be used to trickle-down deposits from higher level accounts to lower level accounts. There may also be mechanisms whereby withdrawals and reservations will trickle-up via special subaccounts in the parent accounts.

## 3.15 Debit and credit allocations

An allocation may be of type debit or credit. By default, allocations are debit-based where credits are deposited in advance and used until they are gone. This might be grants-based or based on a pay first, use later basis. A credit allocation may be used to establish an overdraft buffer or used as a credit account on a use first, pay later basis. Credit-based allocations have a credit limit, supporting a negative balance up to some limit, where subsequent deposits may be made to balance the account.

# 4 Nonfunctional Requirements

## 4.1 Scalability

Our target is the high-end systems for 2006 which we expect to have tens of thousands of processors, thousands of simultaneous jobs and hundreds of simultaneous users. Serialization must be avoided in favor of parallelization and distribution of data and services. Network accesses should be kept to a minimum while using aggregation and compression where possible. The design could consider a distributed approach to help mitigate scalability and fault tolerance issues.

## 4.2 Security

The allocation manager should utilize strong authentication (no clear text passwords) to prevent unauthorized access and support optional data encryption to prevent information from being sniffed. The authentication routines should be modularized to be able to use alternate delegation-based security mechanisms (such as PKI or Kerberos 5) and support the underlying system security infrastructure where possible.

## 4.3 Robustness/Fault Tolerance

Provide some redundancy to avoid a single point of failure. A distributed design could be considered.

## 4.4 Reliability/Fault Recovery

Database performs automatic rollbacks on failed transactions. (Possibly try to do reconciliation allowing post-processing of transactions (withdrawals, reservations) which were allowed to succeed while the database is down).

## 4.5 Verifiability

By providing an option for maintaining the historical state of the bank, previous state can be compared with the transaction history to identify and correct database corruption.

## 4.6 Portability

The allocation manager should be machine and operating system independent wherever possible. It will be developed to a reference Linux platform with the goal of supporting portability to UNIX-based vendor operating systems and architectures, particularly those flavors for which there is a large supercomputing base. It will be written in an architecturally neutral programming language (such as Java).

## 4.7 Heterogeneity

It must support clusters and systems containing nodes with heterogeneous architectures, operating systems and versions, processor number/types/speeds, memory capacity/speed, disk capacity/throughput, swap, network types/throughput, etc.

## 4.8 Logging/Debugging

Consistent, aggregated and "standard format" logging of information
Multiple levels (debugging, information, errors, etc.)

## 4.9 Single point Administration (by multiple parties)

Multiple systems can be administered from single interface. It should be also possible to support accounts nested in a hierarchical arrangement if this would be useful to a large contingency of sites. If a multi-organizational hierarchy was to be supported, it would also be critical to support the ability to delegate management responsibility to the organization which owns the resource.

## 4.10 Performance

As part of the scalability improvements necessary to support thousands of processors, we propose to implement in-memory data-caching for time-critical read-only queries like quotes and balance checks. Also a method to break up large tables would need to be devised in order to ensure good performance for accounting queries.

## 4.11 Modularity

The allocation manager will be written in a modular, object-oriented design making it easier to adapt, update and maintain, in order to adapt to changes in hardware or software requirements.

## 4.12 Usability/Manageability/Ease of use

A web interface with flexible query and update options is envisioned. A command-line interface, both prompt based and individual clients, will precede the web interface so that commands can be scripted.

## 4.13 Interoperability

Using standardized interfaces defined by the Scalable Systems Software Center will promote interoperability, portability and long-term usability. Integrates with batch schedulers and resource management systems.

## 4.14 Extensibility

This system will allow for future change: increase in computer resources, number of users, projects, charging algorithms, currency bases. It will be open source and thus will be able to be modified by the sites to support additional attributes in the accounts, transactions, support new kinds of resources to be managed, etc.

# 5 Software Development Requirements

"Long term maintenance and supportability is of high importance to us"
Lifecycle software development plan

## 5.1 Open Source

In order to be of maximum benefit to the high performance technical community, this software should be open source, allow free distribution, allow sites to make local modifications, customizations and derived works, and promote the sharing of patches, ports, and enhancements from the user community.

## 5.2 Documentation

Proper documentation will be created and made available from a public website. At a minimum, there should be a User Guide, an Administration Guide, an installation Guide, a technical paper, and man pages.

## 5.3 Revision Control

The allocation will be placed under the CVS revision control system.

## 5.4 Test suites

A tests harness will be written for the allocation management system that allows regression testing of its functionality and performance whenever changes are made to the code. It is anticipated to use a test framework like dejaGnu.

## 5.5 Modular design

The code will be written in an object oriented language, with classes for each object type. Additionally, the communication layers will be written as replaceable modules (extension of an abstract/base class) allowing different framing, data-representation and security modules to be selected at runtime.

## 5.6 Packaging

It is anticipated that this will be packaged as a gzipped tarball. It might also be packaged into RPM format.

## 5.7 Installation/Update procedure

It is anticipated that this will utilize the configure, make, make install methodology for installation (or rpm). There should be a mechanism whereby patches may be applied simply, as well as semi-automatic database schema updates between major revisions.

# 6 Interface Requirements

## 6.1 Component Interface

XML Schema validation
Written according to public API to allow easy replacement of components such as the accounting system or the meta-scheduler
Well defined API allows the site to replace or augment individual components as needed

### 6.1.1 Request Types (Transaction Types)

createAccount
queryAccount
modifyAccount
deleteAccount

createUser
queryUser
modifyUser
deleteUser

createMachine
queryMachine
modifyMachine
deleteMachine

createMember
queryMember
modifyMember
deleteMember

createAllocation
queryAllocation
modifyAllocation
deleteAllocation
depositAllocation
withdrawAllocation

transferAllocation
refundAllocation
balanceAllocation

createReservation
queryReservation
modifyReservation
deleteReservation

createQuotation
queryQuotation
deleteQuotation

queryBank
modifyBank
purgeBank
revertBank

createTransaction
queryTransaction
modifyTransaction
deleteTransaction

### 6.1.2 Usage Record Format (Transaction Record Fields)

| **Name** | **Data Type** | **Description** |
|---|---|---|
| Object | String | The object which was operated on, such as account, user, allocation, machine, quotation, etc. |
| Action | String | This field indicates the action performed against the object, such as create, modify, delete, withdraw, transfer, etc. |
| TimeStamp | Timestamp | Time the transaction was recorded |
| AuthName | String | Authorized userid performing the transaction |
| Name | String | "Unique" name/identifier/key indicating the "name" of the object acted upon, such as account, user, allocation, job_id, session_id, reservation_id, quote_id, allocation_id, etc according to context |
| Type | String | Indicates child object type when action involves an association table |
| Account | String | Used to specify project name to which charges are applied when recording resource usage or when account is the child of object |

| | | |
|---|---|---|
| User | String | UNIX/DCE user name specified when recording resource usage or when user is the child of object (like when dealing with user subaccounts) |
| Machine | String | Machine name (This could be a list of machines (systems) for a job which spans clusters and each machine could be a composite name composed of the host, partition, cluster, site, and/or enterprise) |
| StartTime | Timestamp | Time when job starts or allocation/reservation/quotation becomes active |
| EndTime | Timestamp | Time when job ends or allocation/reservation/quotation expires |
| QueueTime | Timestamp | Time when job was initially queued to batch system |
| Wallclock | Int | Wallclock time (how long job ran -- withdrawals) or wallclock limit (max timelimit for job to run -- reservations) in seconds. This time does not include queue wait time, or periods where job is suspended, etc. |
| Processors | Int | Number of processors used (withdrawals) or requested (reservations) by job |
| Nodes | Int | Number of nodes used (withdrawals) or requested (reservations) by job |
| Memory | String | Memory used (withdrawals) or requested (reservations) by job [could be composite of AVG and/or MAX and have K or M qualifiers] |
| Disk | String | Disk storage used (withdrawals) or requested (reservations) by job |
| IO | String | IO Bytes transferred (withdrawals) or IO throughput requested (reservations) |
| Network | Int | Network used (withdrawals) or requested (reservations) by job [could be AVG, TOT, or MAX] |
| Class | String | Class of job (batch, interactive, etc.) |
| Queue | String | Job Queue name |
| JobType | String | Here you could distinguish between RMS job types, NQS, PBS, LSF, LL, etc. |
| NodeType | String | Type of node might factor into performance and charge rate |
| QOS | String | Quality of Service |
| CPUTime | int | CPU Time (for all processes of job) in seconds |

| | | |
|---|---|---|
| ProcConsumptionRate | Float | Percentage of Total CPU used for prorating charge – a decimal number between 0 and 1 |
| ApplicationType | String | Presents a way to categorize use of systems by application type (may be unenforceable and therefore unnecessary) |
| Executable | String | |
| JobName | String | Job or application name |
| Status | String | Completion Status |
| Active | Boolean | May indicate activation or deactivation of a particular user, account, allocation, etc. |
| Amount | Int | Amount debited or credited to account or allocation/reservation/quotation |
| Debit | Boolean | True if this transaction is a debit, false if this is a credit, and null if it is neither |
| Details | String | Addition details of the transaction key,op,value tuples which don't otherwise fit into existing fields |
| Description | String | Description/Reason for transaction |

## 6.2 User/Admin Interface

Should be remotely accessible
Command line followed by Web-based GUI.

## 6.3 Web-based graphical Interface

A web-based GUI will be developed which will give users, managers and administrators a simple interface in which to perform common tasks such as obtaining balance and usage information, making transfers, deposits and refunds, generating bank statements, performing account administration, etc.

## 6.4 API Interface

A public API (Application Programming Interface) will be created which will allow schedulers, meta-schedulers and other allocation managers and services to interface to the bank and make dynamic reservations, withdrawals, quotations, queries, etc.

## 6.5 Protocol Interface

A public wire protocol interface will be developed based on XML according to an SSS standard specification. Other components can communicate directly with the allocation manager over this protocol without having to link in to libraries, modules etc. This will be a flexible request/response syntax which allows for pipelining of requests/responses, and a powerful querying capability supporting the functionality of an SQL backend. The protocol will use support the capacity to validate the XML against the specified XML schema, thereby establishing its comformity to the specification. Validation can be disabled to enhance performance.

## 7 Persistent Data Requirements

### 7.1 Database backend

The allocation manager will take advantage of the powerful querying capabilities of a relational database to store and retrieve the resource utilization information and current balance and state information. This provides concurrency and transactions to prevent data corruption. It also provides better performance than flat-file solutions. Sites can use built-in report utilities or create their own that use bank API's or query the database directly.

### 7.2 Principal tables

| | |
|---|---|
| Account | (manage projects) |
| User | (manage persons) |
| Machine | (manage computing systems) |
| Subaccount | (user_account membership) |
| Allocation | (mapping of resources to individuals, projects, machines and timeframes) |
| Reservation | (temporary holds or pending withdrawals) |
| Quotation | (guaranteed resource quotes) |
| Transaction | (audit_log) |
| Group | (groups of users, machines, or accounts) |
| Bank | (version, bank-wide properties) |

## 8 Preliminary Schedule

| | |
|---|---|
| 1 JUN 2002 | Release initial (V1) XML interface specifications |
| 1 DEC 2002 | QBank adapted to V1 XML interface and security protocols and bundled disabled with SSS_RMS (key must be obtained) |
| 1 FEB 2003 | Release version 2 of the interface specification as well as beta version of new project allocation manager conforming to version 1 of the SSS_RMS interface spec |
| 1 JUN 2003 | TRU64 and AIX initial support |
| 1 DEC 2003 | Release production version of allocation manager, fully integrated and tested with other SSS resource management system components based on V2 of XML interface specs. |
| 1 DEC 2003 | User-oriented problem response system |
| 1 JUN 2004 | Fully integrated with Silver metascheduler |
| 1 DEC 2004 | Fault tolerance supporting 25% cluster loss |
| 1 JUN 2005 | Maintain problem reporting website and mailing lists |
| 1 DEC 2005 | Support parallel checkpoint/restart jobs |

|  |  |
|---|---|
| 1 JUN 2006 | Fault tolerance support loss of 50% of cluster |
| 1 JUN 2006 | Scalability adequate for largest DOE system |

# 9 Validation/Testing Criteria

## 9.1 Test Harness

A test harness will be created and used to perform regression tests on the software so that changes can be quickly verified to not break the code.

## 9.2 List of Reviewers

SDSC – Victor Hazlewood
CHPC – Brian Haymore
ORNL – Stephen Scott
NCSA – Rob Pennington
PNNL – Scott Studham
NERSC – Francesca Verdier